CS 288 2019S Section 102
Hexadecimal Notation

Hexadecimal notation is just another base for representing numbers in, like binary.

In hexadecimal, there are 16 digits: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `A`, `B`, `C`, `D`, `E`, and `F`. The digits `0` through `9` preserve their normal meaning, and `A` through `F` in hexadecimal is `10` through `15` in decimal notation. As any base, we can calculate the value of a given number, say `1CA`, by writing the whole number out extensively. `1CA` in hexadecimal is the same as: $A*16^0 + C*16^1 + 1*16^2$

In decimal, this means: $10*16^0 + 12*16^1 + 1*16^2$

Let's find out what this number is:

```
10 + 12*16 + 16*16 = 10 + 192 + 256 = 458
```

So `1CA` in hexadecimal is the same as `458` in decimal notation.

Hexadecimal notation is useful because it allows you to use powers of two easily without having to write everything out in binary. Converting from hexadecimal to binary is easy: every hexadecimal number is 4 bits, so each hex digit corresponds to a sequence of 4 bits.

To convert a long number, just replace the hexadecimal digits with their corresponding sequence of bits. For example,

```
1CA (hex) = 0001 1100 1010 (bin) = 111001010 (bin)
```

Two hexadecimal digits correspond exactly to one byte, unlike the strange 255 limit in decimal, the hexadecimal byte limit is FF.

In programming languages, a hexadecimal number is usually prefixed with '0x' to make the compiler aware that you are using hexadecimal. For example:

```
0x123 0x0 0xABCDEF 0xCD
```

Assemblers, on the other hand, use the suffix of `'h'`, to match the `'b'` suffix for binary. Most assemblers will also accept the `'0x'` prefix notation, which is generally clearer.

Hex	Binary	Base 10
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Source: OSDev.org